UNIVERSITY OF CANTERBURY

DEPARTMENT OF COMPUTER SCIENCE

HONOURS PROJECT REPORT

BY

COLIN JAMES BURNETT

SUPERVISOR: DR. R.E.M. COOPER

A PROTOCOL FOR AUTOMATIC DATABASE GENERATION

OCTOBER 1981

Abstract

Contemporary database management systems, in restricting themselves to rigid fixed structural representations of the data and their associations, suffer from problems primarily concerning user effectiveness. Recent developments of effectiveness-oriented systems avoid this problem but the approach adopted results in considerable loss of efficiency. By integrating these two approaches in a dynamically controlled structural environment paralleling that of an Entity-Relationship model of the data resource, the two important areas of efficiency and effectiveness are not compromised. Furthermore, additional problems relating to more technical matters such as query optimization no longer arise and apparently conflicting viewpoints on database systems can now be rectified (e.g. the Network and Relational models are special cases of this proposed model). Important analogies in related fields of interest such as distributed databases, knowledge representation and cognitive psychology can lead to new advances and insights into the applicability of this model.

A PROTOCOL FOR AUTOMATIC DATABASE GENERATION

Contents

A PROTOCOL FOR AUTOMATIC DATABASE GENERATION

1. Introduction

> The only constant thing in today's world is change.
>
> Anon.

The field of database systems has been an active area of research in Computer Science for more than ten years. Nevertheless, a unanimous definition of the term "database" has still not been achieved. GUIDE/SHARE defines a database as:

> "a named collection of units of physical data which are related to each other in a specified manner " [11]

which, although a rather general definition, seems to encompass the overall idea of database systems. Henceforth, within the scope of this report, the term "database" shall be used to denote the data and associations being modelled on a computer system. Adding control over the structure and access to the data results in "database systems", with the control of the structure of these database systems being managed by "database management systems". In contrast to this pragmatic view, the term "data resource" will be used to refer to the actual data elements and associations as they exist in the real world. However, these definitions provoke the same criticisms as the GUIDE/SHARE exposition. This generality seems to arise from the conflicting objectives to which database systems are addressed.

The provision of more relevant information to the user is perhaps the most important of these, and is usually done by representing more relationships between the elements of the database than are currently required. In this way, users are able to extract combinations of data which were not originally conceived when the database was designed. Related to this desire for the provision of more information is the idea of corporate or central control of the data resource. A database system provides a convenient way in which the operations of a corporation can be monitored and perhaps even integrated. This avoids the situation where two departments are engaged in the same development activity, or worse still, are working at cross purposes. These problems of redundancy and inconsistency also have considerable bearing on the integrity of the data resource. In the past these objectives have usually been met by the construction of systems based on one of three database models.

The first of these to appear was the hierarchical model as illustrated by the IMS database management system first released by IBM in 1971. This model attempted to simulate organizational structures by grouping data elements into hierarchies. This met with partial success, although, some applications did not possess the necessary hierarchical view of the data and, in many of these cases, the model was not sufficiently powerful to represent accurately the structure of the data. This inadequacy, particularly concerning the representation of networks, has since caused the hierarchical model to fall out of favour and today the

model has all but disappeared (although it is still used to model organizational structures).

Soon after the release of IMS, the TOTAL database package was released by CINCOM (1972). This system took a different approach to that of IMS, namely the network approach. This was suggested by the CODASYL DBTG in its report produced in 1971 and has been in vogue for the past ten years. In the network model, data elements are grouped into logical hierarchical structures termed sets. Sets are one-many data structures with one owner and an arbitrary number of members. As it is possible for members of one set to be owners of another set and vice versa, these sets can be interwoven at will to allow a true network structure.

Also, now equally popular, is the relational model which stands on a firm mathematical basis. This has been likened to a flat form representation of the network model and possesses similar descriptive power although it does have the advantages of easier query specification and greater data independence.

2. Important Objectives in Database Systems

> I know of no way of judging the future, but by the past.
>
> Patrick Henry.

2.1 Efficiency

2.1.1 Contemporary Models

The hierarchical and network models were designed from the point of view of machine efficiency and consequently the user interface to the database system suffers from the awkward data structures and access methods imposed. This emphasis on efficiency stems from the hardware-software cost trade-offs apparent around the early 1970's. Computer hardware was still fairly expensive and typically comprised about 80% of the total cost of a computer system. Under these circumstances it is hardly surprising that machine efficiency assumed paramount importance.

The relational philosophy is a more unusual case because of the way in which it came into being. The idea was first presented by Codd in 1969 in an attempt to describe database systems formally using set theory. However, when attempts were made to implement the philosophy, the systems were in general too slow to be practical. Thereafter all attempts at feasible relational database systems have used some form of additional access method(s), often transparent to the user.

2.1.2 Access Methods

Such access methods, for both the relational and the other two models, normally fall into a surprisingly small number of categories. The simplest and slowest of these is the Sequential Search [13,p393] where the entire file is searched sequentially until either the required element is found or the end of the file is reached. While this is the slowest access method, it is the least expensive in terms of additional storage and maintenance overheads.

The Sequential Search makes no *a priori* assumptions about the structure of the data which is why its access time is so poor. The Binary Search [13,p393] has substantially faster access time because it assumes that the file is sorted in some predetermined order, and it uses this knowledge to decide where to probe for the data. To satisfy the assumption that the file is sorted does mean that update times can be very long for this search however. Still faster access times can be achieved if we know the range of values for the keys and can assume that the keys are uniformly randomly distributed. The Interpolation Search [13,p416] (Appendix A) makes use of this additional knowledge to achieve this result.

An extension to this idea of extracting information from the actual data to determine how best to access it is used in what I have chosen to call the Regression Search (Appendix A). A

simpler version of this approach attempts to reconcile the three important parameters under consideration - space, access time, and update overheads. The Index Sequential Access Method or ISAM divides the elements of a file into a number of blocks by allocating the elements to blocks according to some predefined collating sequence (commonly alphabetic). An index file can then be created for this file and, as the allocation strategy is known, the appropriate block can be retrieved and searched sequentially.

ISAM incurs very high update costs when blocks become full, so a more flexible structure like Key Sequential Access Methods (KSAM) is required for volatile applications [13,p451]. Here, the index file is maintained as a balanced tree structure. Each node of the tree contains one or more pairs of key and pointer values. This structure is more flexible than the ISAM index because additional nodes can be added anywhere in the tree by extending its depth. In situations where change is the norm, this method still incurs considerable overheads in both update and access and a different approach may have to be made to the problem.

Hashing [13,p506] avoids these troublesome overheads by victimizing storage resources typically using only about 40% of the allocated storage. The basic principle is to map the range of key values (which is usually considerably larger than the actual number of keys) into a much smaller range, and create an index file based on this smaller range.

All of the techniques discussed so far work equally well from both the view of initial access into the database as well as a means of getting from one file to another from inside the database system. Moreover, this latter type of access can make use of two additional access methods that are impractical from the user's point of view.

The first of these, simply called Direct Access, relates to the concept of a record in contemporary database systems. Getting from one field to another in the same record can be thought of as an access method with trivial access time, minimal update overhead, and (usually) no additional storage. While this may seem to be the ideal situation, problems occur when the relationships between fields are not in a one-one correspondence. For instance, if students can take a variable number of courses then we need to allow sufficient space for the maximum number possible and, in this way, considerable space can be wasted. Also as the number of attributes stored in each record is increased, the number of records that can be read in from secondary storage at one time is reduced, and this can increase the access time.

Variable associations between elements can be represented without wastage of space if Pointers are used (the other internal access method). This approach incurs greater access and update times and uses a small amount of additional storage to contain the pointers.

The above brief descriptions of what I consider to be the seven types of access method are summarized below under the headings of: Usage, Access Method, Update Time, and Storage Requirement. However, it must be remembered that these ratings, while only estimates here, can be algebraically calculated to produce better results. Nevertheless a simple rating scheme will suffice for the purposes of this report.

access method	usage	access time	update time	storage requirement
Sequential	internal/user	10	1	0
Binary/Interpolation	internal/user	5	8	0
ISAM/Regression	internal/user	3	10	3
KSAM	internal/user	4	5	5
Hashing	internal/user	2	4	10
Direct	internal	0	1	0
Pointers	internal	1	2	1

Fig 1. - Comparison of Access Method categories in terms of estimated performance.

2.2 Effectiveness

2.2.1 User-Oriented Systems

Effectiveness is a measure of how well a product suits the needs and desires of the user. For database systems, this relates both to the ease of use and to the facilities provided. With the hardware-software cost trade-offs mentioned in the previous section being all but reversed there is an increasing demand for such systems to be more "user-oriented". This has led some firms into the development of packages which address themselves primarily to this objective, with the efficiency criterion typically being relegated to a poor second. These systems are usually "add-ons" to an already existing DBMS and usually result in a rather clumsy interface because of these apparently conflicting objectives. Nevertheless, the results of this approach certainly seem to be reaping considerable benefits (e.g. Burroughs LINC [10]) despite this internal incompatibility.

2.2.2 The Conceptual Schema

This user-orientation aspect of database systems was recommended by ~~ANSI/X3/SPARC [2,19] in 1972~~ recommended by ANSI/X3/SPARC [2,19] in 1972 where they suggested splitting the schema into three parts - the external, the conceptual and the internal schema. Effectiveness-oriented systems normally treat the already existing DBMS schema as the internal schema and use an enterprise view [18] of the organization's data resource as

the conceptual schema. This enterprise view is usually based on some dialect of the Entity-Relationship model as proposed by Chen [4] and actively researched by many others [15,17,18,21,23].

2.2.3 The Entity-Relationship Model

The E-R model is based on two fundamental concepts: the entity and the relationship. An entity can be defined as:

> "an abstract or physical quantity which is an element of the data resource being modelled"

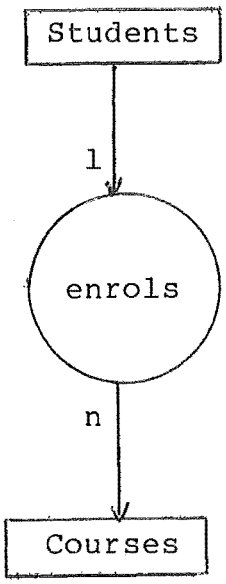
or more simply as:

> "a noun describing an object of interest to the user".

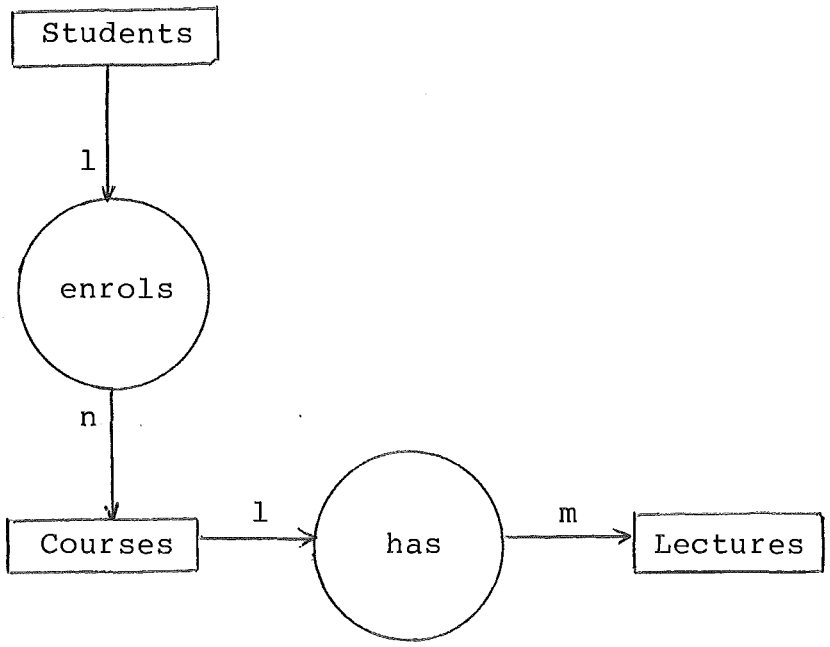
Conceptually, this particular aspect of the model is easy to understand; however, difficulties arise concerning the representation of relationships in a computer system. Relationships come in various shapes and sizes and, unfortunately, there are a very large number of different types. They differ in both the number of entities which they relate and in the semantic (meaningful) constraints they impose. This can make things very difficult to handle, however, a novel approach to this problem reduces the complexity by breaking down the relationships into a small set of primitive relations and created "events" (entities that correspond to activities such as "the act of selling") [10,18,21]. Similar to the syntactical derivation of an English sentence, the complex relationships can be reduced

to relationships which have only one link to the subject of the relation, one link to its object, and possibly one further link to additional events that each "describe" an indirect object that may exist. (see also Ridwan [18,p100]). This solves the problem of a variable number of associations but the semantic constraint difficulty has yet to have a satisfactory solution. An example will best clarify this approach.

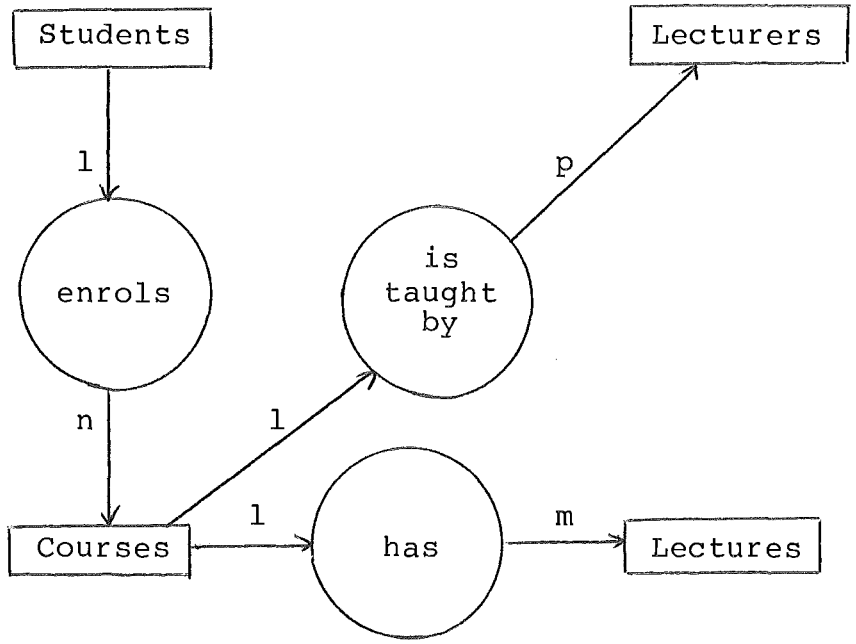
1. Each student enrols in n courses.



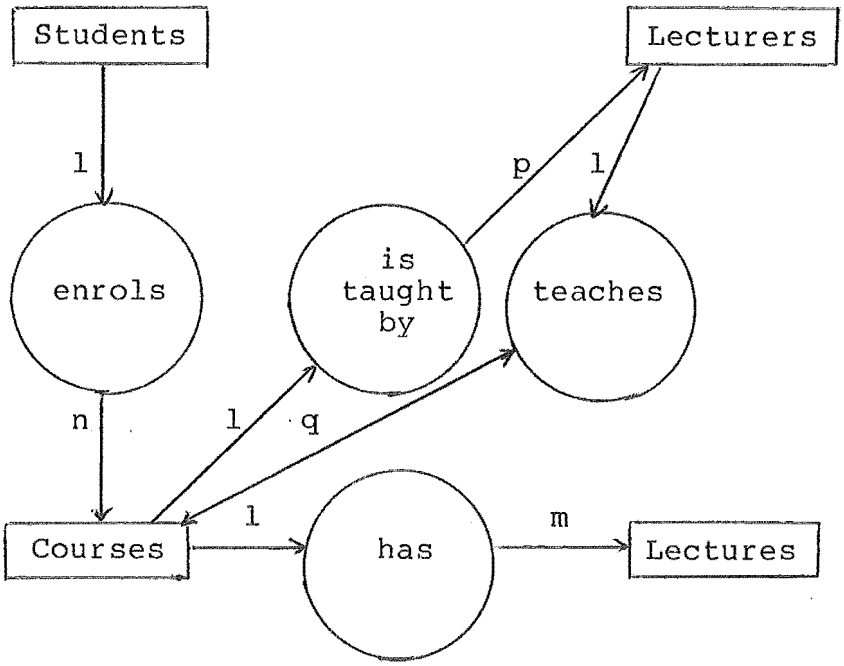
2. Each course has m lectures.



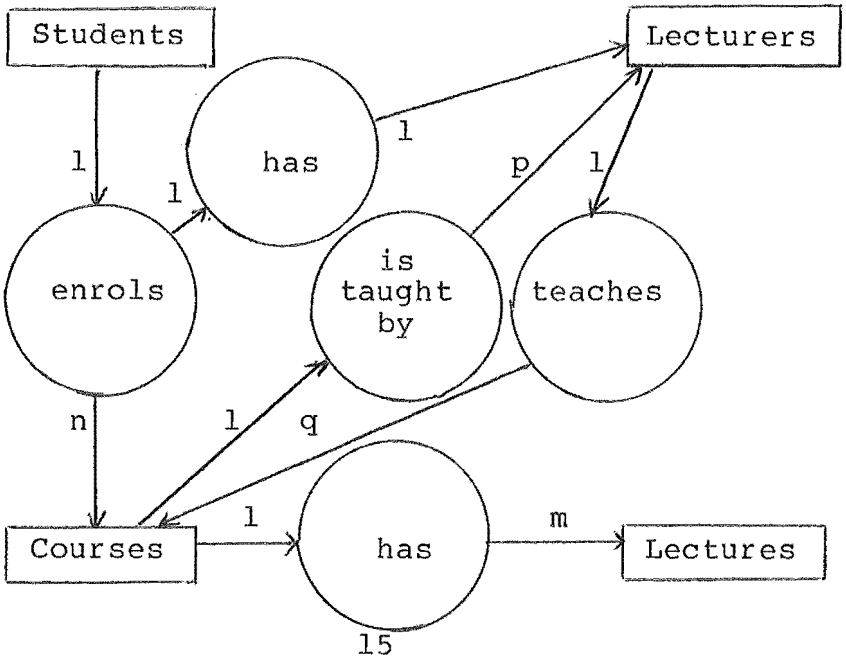
3. Each course is taught by p lecturers.



4. Lecturers teach q courses each.



5. Each student enrolled in a course is taught by the same lecturer for that course.



In these examples, the nouns enclosed in rectangles signify entity classes, with the circled verbs denoting events. Often these events are trivial (e.g. "has" in diagram 5 which has no associated information) but, at the very least, they indicate the correspondence between the related entities and events. In the diagrams, this information is indicated by the numbers associated with each of the links and is referred to as the cardinality of mapping [18,p63]. This concept can be extended to handle variable associations based on probability distributions so that restrictions like: "each student can be enrolled in no more than n courses" can be represented accurately. Another point worth noting is the directional aspect of the relationships. This is contrary to the generally accepted definiton of a relationship. However, this approach not only enhances the clarity of the model, but also removes any ambiguity that may arise with many-to-many relationships (see the lecturer-course relationship in the fourth diagram). Another reason for restricting the model to directional relationships concerns the scope of the database system being implemented. On the boundaries of the model domain, it is often not desirable, and in places not possible, to represent bidirectional relationships.

2.3 Flexibility

To my mind, the third major objective of database systems is flexibility. This differs from the previous two goals because it is not concerned with the representation of the data but with the change in the representation of the data. Information requirements are influenced by external stimuli and growth of the organization may expand the corporate data resource. It must be possible to handle both of these situations if a database system is to be considered as a long-term proposal. Contemporary database systems do not perform well on these aspects, with variations in information requirements necessitating manual "tuning" of the system by the database administrator, and expansion of the data resource requiring a manual rebuild of the database. This situation is now being realized and work by Stocker and Dearnley on evolutionary database systems [6,20] is an attempt to introduce dynamic control into database systems to solve these problems. However because of significant overheads in monitoring and reorganization, their evolutionary model has been restricted to batch systems.

3. A Protocol for Database Systems

> If you restrict yourself in advance to any fixed technique, you create a handicap for yourself that a person with all the options does not have.
>
> T.M. Georges [7].

3.1 Underlying Concepts

3.1.1 Satisfying the Major Objectives

When comparing the three main objectives of efficiency, effectiveness and flexibility in the light of contemporary models, there appears to be a trade-off between them. By adding a conceptual schema to an already existing DBMS, loss of efficiency is incurred. On the other hand, if we try to increase the flexibility of the system, we arrive at an awkward and hence difficult to use conceptual schema, or else a severly restricted DBMS, both of whch are detrimental to the effectiveness of the system. It would appear, therefore, that what we require is a suitable cost trade-off between these three desirable characteristics. However, this conclusion assumes that the underlying concepts of contemporary database systems are the best foundations for database systems.

Careful examination of the three previous sections reveals that the three objectives can in fact be fully integrated, even to the point of actually complementing rather than competing with

each other. This surprising result can be seen very simply by drawing analogies between entity classes and files, and between associations and access methods. In other words, I am saying that an E-R model along the lines previously mentioned can be implemented directly by taking advantage of these analogies. (internal schema = conceptual schema). Furthermore, the analogy between relationships and access methods has considerable scope for variation; a fact which can be put to good use in improving the efficiency and flexibility of a database system.

3.1.2 Access Paths

The concept of an access path is particularly important to the achievement of this aim. The representation of a relationship in the computer is an access method between two files, according to this model. While this is how it is physically implemented, it is often easier to view relationships as directional paths from one file (entity class) to another. Throughout the rest of this report, this abstract view will be assumed and the directional paths will be referred to as access paths. Using this view of access paths we can look at how improvements in efficiency can be achieved.

3.1.3 Dynamic Control

To do this, I shall introduce the concept of dynamic control into the database system. Notice here that the internal representation of the data resource is an analogous representation of the true data resource because of the enterprise view approach adopted. Therefore changes in the information requirements of the organization will be reflected in increased activity in the corresponding entity classes (files) and relationships (access methods). Dynamic control means using this information on change in activity to perform some form of optimization on these access paths to make the more frequently accessed data easier and faster to retrieve. Obviously, this form of optimization is highly volatile as information requirements can change rapidly, hence the need for an evolutionary model. To enable evolution to be performed appropriately on a particular database system, it is necessary to not only accumulate statistics on the usage of access paths but also to undertake some form of forecasting. The accumulation of statistics is conceptually very simple, requiring only some measure of how frequently the access path is used. Actual implementation is more difficult as the "frequency" of access is a hard concept to define. Also, any forecasting technique used has to be particularly light on resource requirements as most database systems will have many access paths.

3.1.4 The Need for Forecasting

Exponential smoothing [3,22] is a forecasting technique which maintains a "level of activation" measure for each access path, which is updated at discrete time intervals according to the following formula:

$$R = ALPHA * X + (1-ALPHA) * R$$

where R is the "level of activation", X is the amount of activity on this path since the beginning of this time interval, and ALPHA is a value between 0 and 1 which controls the effects of random variations in the measured activity level. This technique requires only three storage locations per access path - one for the "level of activation", one for accumulating usage since the last time interval, and one for ALPHA. The storage of a separate ALPHA for each access path is a questionable decision as it assumes that volatility is a property of access paths. (Notice that the larger ALPHA is, the more rapidly the access path responds to change, be it random or significant.) On the other hand, information retrieval systems appear to have a low structural volatility whereas knowledge representation systems seem to have an inherently high volatility. So perhaps volatility is a property of the database system and, in this case, only one value of ALPHA needs to be stored. This decision concerning volatility can only be decided after simulating actual database systems. An alternative forecasting strategy which provides better response to change is presented in Appendix B.

3.1.5 Global Considerations

Assuming, therefore, that we have a technique for predicting usage of access paths, how do we go about using this information? As was mentioned in the previous section on efficiency, it is possible to calculate the performance of access methods for access time, update time, and storage requirement theoretically. Surely all we have to do is fit the best access method to each of the access paths and the problem is solved. Unfortunately, this is not the case as we have neglected the global objectives and constraints associated with database systems. Each of the three performance measures (access time, update time, and storage requirement) can take on the form of either an objective (goal) or a constraint (restriction). For instance, it is usual for storage to be a constraint and access time to be an objective. However, in some systems there may be much contention for storage and in these circumstances storage requirement could perhaps best be expressed as an objective. Multiple objectives can be resolved using the technique of proportional representation in goal programming [5,p79], where a trade-off function is specified between the objectives, and each is represented in the objective function according to its importance. Access time and update time pose somewhat of a problem as true access and update may require the traversal of more than one access path. Therefore, I will assume here that specifying either of these parameters as constraints implies that the maximum refers to each of the access

paths in the database system, whereas specifying either as an objective will refer to the total access or update time resource for the whole database. These global considerations are established when the database system is first generated, and cannot be altered without regenerating the entire system.

3.1.6 Generation or Regeneration?

Global objectives and constraints differ from the previously discussed aspects of the system in that they are concerned with the generation of the system instead of its evolutionary behaviour. While it is true to say that database evolution can take place as a regeneration of the entire system, such an approach is rarely feasible. Therefore, for the purposes of this discussion, generation will be considered as a special case of evolution and not vice versa as is usually the case.

The reason for this curious stance stems from the overheads associated with establishing an optimal system at either of these stages. Considerable work has been done on optimal generation of database systems [14] based on the assumption that this is a once-only task, and hence the significant overheads are still warranted. However, the model I am proposing here is evolutionary and therefore the use of this approach is prohibitively expensive. Therefore, I propose to abandon the quest for optimality and, instead, aim for the more realistic target of a "good" database system.

3.1.7 The Heuristic Approach

This approach lends itself nicely to heuristic methods of solution.

> "Heuristic problem solving involves the development of a set of rules called heuristics, which hopefully will aid in the discovery of one or more satisfactory solutions to a specific problem. The emphasis is on satisfactory - there is no guarantee of optimality" [5,p511].

An insight into how these heuristics can be developed may be obtained from examining the optimal solution procedure. The problem as it stands is an example of the Knapsack problem in Operations Research [5,p236]. The problem is characterized by having a number of potentially useful items (access methods), each of which utilizes a specific amount of some scarce resources (the global constraints) and each worth a certain amount which we wish to maximize (the global objectives). Like most useful algorithms in OR, the optimal solution to the Knapsack problem can be found by solving it as a mixed integer programming problem, a solution technique which is, unfortunately, very heavy on resources.

A more practical solution is to assume that local optimization will still produce "good" systems. Local optimization consists of taking the current solution and perturbing it by actively altering the access path with the most

potential cost savings, while the alteration of any additional access paths is made only to maintain the feasibility of the solution. An informal outline of a possible heuristic to perform this function is presented in Appendix C. Considerable advantages are gained from the use of these heuristic techniques, the most obvious one being reduced overhead in supervising the evolutionary role of the database system. Typically these "evolutionary runs" would occur at fairly infrequent intervals measured in terms of weeks or months anyway, so reasonable overheads in running the heuristic can be tolerated. A further benefit to be gained concerns the generation of the initial system. As its behaviour is evolutionary, "optimal" systems would only ever be short-lived, so the same heuristic can be used to initially generate a "good" system by starting with a database where all the access paths have the cheapest access method assigned to them, and then applying the heuristic iteratively until no more improvement in the solution occurs.

3.1.8 Implementation Considerations

Conceptually, this idea is very appealing, but some problems do arise regarding its implementation. The only major one I have been able to determine so far concerns the access methods and where they come from. It is all very well to say: "convert access method i into access method j", but these access methods all have to be stored somewhere, and while there are only seven important types of access methods, some of these types can have a

very high number of possible variations. The suggested solution is to have seven access method generators which can be passed parameters which describe the access method required. These access method generators can be stored off-line as they would only be required when the heuristic is run; the access methods produced are incorporated into the database systems.

3.1.9 Analogies in Related Fields

Many of the concepts incorporated into the proposed model have been realized in fields related in a greater or lesser extent to database systems. Albano and Orsini [1,p91] state:

> "Recent works on languages for modeling complex database application environments show overlapping issues with other research areas such as Artificial Intelligence and Programming Languages"

and, writing about languages for modelling databases and tools concerned with their implementation, say:

> "The tools and the language must be strongly integrated in a unique interactive programming system in order to provide an effective user-designer environment. This coexistence appears as fundamental in those applications which require frequent maintenance to adapt the database to the new needs and to accomodate changes, but it is also important to design and debug complex applications that are relatively static".

Meanwhile, Hendrix [9,p132], while talking about his KR project, KLAUS, has this to say:

> "By telling KLAUS how the database is interconnected logically with various aspects of the real world that have been communicated to KLAUS previously, KLAUS may relate the utility of the database to users who know only about the real application and who do not understand the underlying DBMS technology and encoding schemes and conventions".

The field of Knowledge Representation was also quick to catch on to the concept of access path optimization which is fundamental to this protocol. Wiederhold, in talking about access path optimization for his KBMS says:

> "A responsive database should be self-organizing; that is, the decision as to which auxiliary files and access paths should exist, and how they should be represented, should be made by a database management system, with the user interface remaining invariant. The structural model provides a basis for a consistent interface, while permitting many alternative physical implementations. Use of dynamically changing statistical knowledge of the activity applied to the file structures can then improve the systemwide performance. Hence certain of the database administrator's traditional functions in database

> tuning may be partially assumed by the system" [23,p35].

Katz [12,p136] also has something to say on access path optimization:

> "Physical design proceeds by assigning properties to logical access paths (derived from relationships) which ensure that the most heavily travelled paths receive the most favorable combinations. This assignment is used as a specification for realizing a physical database schema for a particular model and system".

In Cognitive Psychology, a paper by Hayes-Roth and Hayes-Roth [8] outlines an Adaptive Network Model for human long-term memory which proposes that the access time to any particular concept is related to both how well it is learned, and how frequently it is accessed.

3.2 The Generation Phase

In this section I will present the steps that are required for the generation of a database system using the protocol outlined in the previous section. These steps are only preliminary however, as much of the detail of the protocol will not become apparent until it has been theoretically modelled or a prototype system implemented.

1. Perform systems analysis to determine whether or not a database system is an appropriate solution.

2. Determine the scope of the data resource that is to be modelled in the computer system. This involves a clear specification of the areas of an organization that are to be modelled so that the appropriate section of the corporate data resource can be formally specified for the database system.

3. Create an Entity-Relationship model of this section of the data resource by identifying the entity classes and the relationships, and normalizing the resulting model so that all relationships are directional and the maximum number of associated relationships for an event is three (see the section on effectiveness for a more detailed discussion).

4. Append values to each of the directional access paths to indicate the cardinality of mapping. If some of these mappings are variable then a probability distribution will be required of which the Beta distribution appears to be the most appropriate and the easiest to work with.

5. Describe the type and range of values for each entity class and non-trivial association. Entities are assumed to be characterized into entity classes so that their type and range correspond to that of the class. Non-trivial associations are those for which information about the association is to be stored. Trivial associations have no such information and appear only implicitly in the physical implementation of the model.

6. Supply initial estimates of the number of entities in each of the entity classes and non-trivial associations. This information is used to determine the storage requirement for the file, and also for calculation of the conversion cost between different access methods. In actual implementations this step may be unnecessary as the DBMS would have all the data stored and hence could calculate it exactly. However as the data collection phase is often expensive and time consuming, initial estimates may help to justify any decision.

7. Give educated guesses on the initial activity levels for each of the access paths. This step is not crucial as the database system will adapt itself over time, but supplying estimates of the activity levels will result in a better initial system.

8. Specify the implementation parameters for the database system. These will include setting the value of ALPHA for the forecasting routine and establishing the time between runs of the heuristic. It is conceivable that future development of this protocol will allow automatic control of these parameters too, requiring only initial estimates for these values.

9. Formulate the global objectives and constraints. This step involves establishing trade-off functions between multiple objectives and specifying limits for the global constraints. The three parameters which can be controlled in this manner are the total storage requirement, the access time for either the whole system or the individual access paths, and the update time for similar situations.

10. Assign the cheapest access method to each of the access paths. The actual access method and the feasibility of the initial solution should be irrelevant to the operation of the heuristic.

11. Initiate execution of the heuristic.

12. If no feasible solution results then relax the global criteria and rerun the heuristic until either an acceptable, feasible solution is found or the method is abandoned.

3.3 The Evolution Phase

Four different types of evolutionary behaviour which affect this protocol have been identified. The most frequent and most obvious of these occurs when the information requirements of the user change. For this there are only two steps as the system is largely automatic.

1. If the access method generators and/or the heuristic procedure are stored off-line then load them into the machine.

2. Initiate the execution of the heuristic.

If the first step is not required (i.e. the DBMS is on-line) then the entire procedure can be completely automated.

The next most likely change to occur is in the environment itself. Through either the growth or restructuring of the organization, the data resource may change and, if the section currently being modelled by a database system alters, then the whole system must be regenerated from the beginning.

Thirdly, it is conceivable that over a period of time, the information requirements may change so significantly as to affect the characteristics of the entire database system (e.g. volatility). At present, these characteristics are assumed to be under manual control. However, if the heuristic is designed well then these parameters could be automatically controlled. Either

way, there is no need for a total regeneration and the heuristic need only be executed as a normal evolutionary run (although it may take a little longer to run).

The fourth possibility for evolution concerns the database management system itself. Improved techniques and access methods are certain to be developed and, if designed wisely, the DBMS should be able to take advantage of these discoveries by incorporating these additional access methods, thus gaining the corresponding increases in performance.

3.4 Potential Problems

Currently, the largest unknown factor in the development of this protocol concerns the suitability of heuristic techniques. Unfortunately, this cannot be determined without either a thorough theoretical analysis or a prototype implementation.

Even if a suitable heuristic can be found which gives satisfactory performance, the overheads introduced by either the statistical procedures or the heuristic itself may be too large for this approach to be feasible. Again, this can only be determined by simulation or theoretical analysis.

Considering now the users' side of this proposal, there has been some doubt as to the suitability of the E-R model as an accurate representation of the corporate data resource. Possible anomalies in the model may arise in complex situations which it may only be possible to avoid by the introduction of semantic constraints.

3.5 Possible Implications

Apart from the design goals of more user effectiveness, greater flexibility and increased efficiency, the proposed approach is also conceptually very simple and adaptable to improvements in the field of database systems.

Its flexible data structuring arrangement allows it to assume the structures associated with the three contemporary models (Hierarchical, Network, and Relational), however, in practice this is not very likely to occur as the "best" arrangement is probably some other combination of the files and access methods.

The similarity of this structure to the actual data resource actually provides considerable benefits. For a start, there is no need for a specially trained database administrator; the techniques should be simple enough for the user to learn.

Also, as requests for information are usually formulated in the context of the corporate data resource, the development of a powerful and easy to use query language should not be too difficult.

Query optimization comes under scrutiny as well. Surely if the query was to be performed a great many times, then accessing the information analogously to how it would be obtained in a manual system will simply improve the performance of the corresponding access paths, effectively optimizing the query.

A PROTOCOL FOR AUTOMATIC DATABASE GENERATION

Distributed database systems pose several difficult problems for contemporary systems but a simple analogy between the nodes of a network and the entity classes (files), and between the communications links and the access paths (access methods) reveals that the distributed approach is basically a special case of the E-R structure where the access paths have a slower traversal time.

Finally, if indeed this is a reasonable simulation of human long-term memory structure, then insights into how we store and process information may be gained to the benefit of not only the field of Database Systems, but also to the fields of Artificial Intelligence, Cognitive Science, and Cognitive Psychology.

4. Suggestions for Further Research

> Nothing is worth looking at which is seen either too obviously or with too much difficulty. Nothing is worth doing or well done which is not done fairly easily.
>
> Samuel Butler.

Although the outline of a suitable heuristic procedure has been given in Appendix C, there is still much work to be done in developing a formal heuristic to perform this evolutionary function. In essence, the suitability of this heuristic and its performance capabilities is the crux of the proposed system, and much care and thought would have to be put into its development.

After the heuristic has been developed, or even as part of its development, a theoretical analysis of the model could be performed. While this step is not essential to the realization of a working system, it does highlight potential problem areas in the design and provide a more formal description of the model.

The construction of a prototype system represents the next step in the evolution of this model and, aside from testing the assumptions on which the model is based, it allows the more difficult design decisions (e.g. the scope of structural volatility) to be solved by simulation.

Either as a further development from this step, or as a separate area of research, the simulation and testing of this

approach as a theory to how human long-term memory is structured has many exciting possibilities and implications.

5. Conclusions

I have presented here a novel approach to the problems surrounding database systems and, as the result of this new approach as to how database systems can be structured, suggested a protocol for the automatic generation and regeneration of database systems. The adoption of this approach seems to alleviate most of the major problems concerning database systems and, apart from some minor technical problems relating to the implementation, seems simple and effective to use.

The potential for these advantages appears to arise because of the much greater flexibility of this model compared to the stringent structural restrictions existing for contemporary database management systems.

Appendix A: The Regression Search

The development of this search arose from applying the concepts used in the proposed model for database systems to the searching problem. Quite simply, the Regression Search extracts information from the data being stored and hence has a better idea of where to locate it when the data is later accessed.

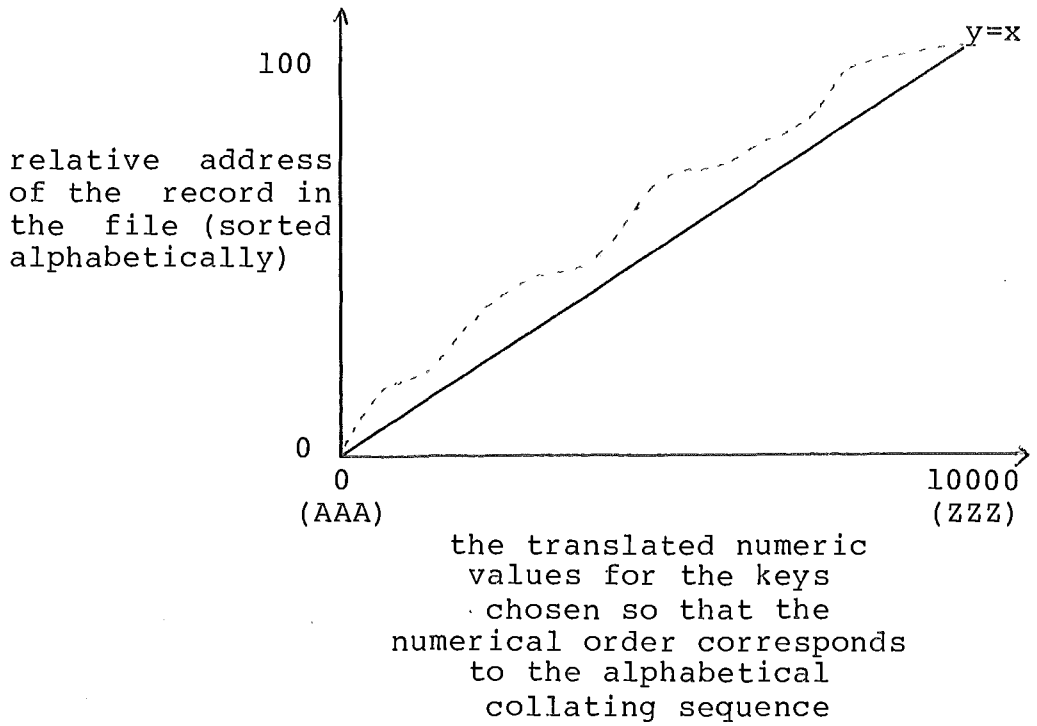
The Regression Search is restricted to sorted files, it uses a small amout of additional storage, incurs a nominal update overhead, and provides very fast access. Its access performance is not as good as that of hashing, however it uses considerably less additional storage. At the other extreme, the Binary Search uses less storage but is substantially slower at accessing the data. The Regression Search, like ISAM, lies somewhere in the middle, however evidence suggests that the Regression Search will perform better than ISAM given equivalent amounts of additional storage.

The best desciption of the Regression Search can be made by comparing it to the Interpolation Search [16]. The Interpolation Search is a modified form of the Binary Search where each of the probes is selected as a "guestimate" instead of the midpoint of the remaining solution space to be searched. For a better idea of how this works, imagine looking Bloggs up in a telephone book. No-one in their right mind would start by turning to the middle of the book but instead would turn to where he or she expects the B's to be. This is the principle behind the Interpolation Search

which makes the rather coarse assumption that all the keys are uniformly randomly distributed (just as many B's as there are X's).

Of course, after making the initial guess, some form of iterative procedure can be used by deciding from the value of the key retrieved which of the two sections of the solution space the key may lie in, recomputing the new range of values, and having another go at it. This works fine until you get near to the end of the search where it is possible to keep accessing the same element repeatedly. To avoid this problem when it happens, a Sequential Search procedure is used at the final stage of the search. This has the disadvantage of giving the Interpolation Search a worst-case time of O(n), however, the expected access time is only O(log log n) [16].

The Interpolation Search can also be characterized by a simple graphical approach. Assume that you have a function that translates your keys into positive numerical values while still preserving their order, and let these values be points on the x axis. Now make the points on the y axis correspond to a linear ordering of the positions in the file. For every element in the file, plot the point indicated by the co-ordinates (x,y). The result should be a function with a non-negative first derivative. (If not, then your translation function does not convert the keys into numbers which are in the same order as the keys are in the file). The Interpolation Search assumes that this resulting function is y=x.



In practice, however, this assumption is usually invalid (c.f. telephone book), and the dotted line is more representative of actual functions.

Regression Search extends the concept by allowing the mapping function to be determined by curve fitting, usually with some polynomial regression technique. Under this situation, the key is first translated into its corresponding numerical value (hashing), and then plugged into the mapping function to arrive at the initial probe into the file. Technical problems concerning additional iterations of the procedure are more complex than for the Interpolation Search, nevertheless a suitable technique has been derived which is too detailed to present here.

The price incurred in using a Regression Search is the extra storage requirement (3n+2) for the regression matrix of order n, and the update of these elements whenever the file is modified. In return you get significantly faster access times. (In a simulation experiment carried out on a file of 1,000 street names and 10,000 accesses, Cubic Regression performed more than 3 times better than Binary Search).

No theoretical analysis of the Regression Search has yet been performed and more extensive simulation runs still have to be conducted to further develop this method.

Appendix B: Event-Oriented Forecasting

The basic exponential smoothing technique is a simple and effective way of indicating the activity of an access path, however it is not so good at predicting future activity. Double and triple exponential smoothing [22] are extensions to this basic method which incorporate trend analysis into the predictive capabilities of this approach.

Unfortunately, these techniques are all discrete in the way they measure the activity levels. (i.e. they accumulate the number of accesses over a certain time period and then report the results at the end of that period). This means that sudden changes in the activity level of an access path will not be reported until the end of the current time period, and then will have their effect significantly reduced by the scaling factor ALPHA. In this context, the time period performs an additional smoothing effect which, for the user to reduce, requires introducing much more forecasting overheads. (shorter time periods).

A more responsive system could perhaps be designed in an event-activated way, where the activity levels are not updated until either the access path is used or the activity level for that path is requested. The storage location for accumulating the number of accesses associated with an access path is no longer required. Instead, the time of the last update or access to the activity level must be kept. The formula for this

event-oriented exponential smoothing will resemble:

```
R = (1/(CURRENTTIME-LASTEVENTTIME))*R + ALPHA*(M-R)
```

where R is the present activity level, M is the maximum activity level, and ALPHA is a scale factor between 0 and 1. The last term of this formula is ignored when retrieving the activity level. Much more research has still to be done on this idea.

Appendix C: A Suggested Heuristic

In this appendix, I shall describe, informally, a possible heuristic to perform the generation and evolution phases of the proposed model.

The heuristic examines every access path and looks at the effect of converting the current access method into each of the other six types. This effect is estimated by the following formula:

$$C = CC(N,M) + GOAT*(MCAT(DAT)*FAT*PH) + GOUT*MCUT(DUT)*FUT*PH) + GOS*MCS(DS)$$

where: CC(N,M) is the conversion cost from the current access method to the proposed access method and is a function of the two file sizes, N and M, being related

GOAT is a zero-one variable; one if access time is a global objective, and zero otherwise

MCAT is the marginal cost function for access time

DAT is the difference in access time between the two methods

FAT is the forecasted access time requirement

PH is the planning horizon length

GOUT is a zero-one variable; one if update time is a global objective, and zero otherwise

MCUT is the marginal cost function for update time

DUT is the difference in update time between the two

methods

FUT is the forecasted update time requirement

GOS is a zero-one variable; one if storage is a global objective, and zero otherwise

MCS is the marginal cost function for storage

DS is the difference in the storage between the two methods

The conversion which results in the smallest value for this function is further considered as a possible candidate for evolution provided this value is negative (cost savings incurred). Usually, one or more of the global constraints is violated by simply performing the conversion and if this is the case, we have to examine the possibility of perturbing other access paths. If either access or update time is a global constraint, then the feasibility of the solution can be checked immediately as these relate directly to the access path. However, if storage is a global constraint which is violated then we have to select the current access methods which give the least return per unit storage. We consider perturbing these by adding their resulting costs (calculated from the formula and usually positive) to the minimum value of the function. If the value is still negative and the feasibility condition is satisfied, then the change is made. This procedure is repeated until no more changes occur.

The marginal cost functions used in the formula are re-evaluated every time a perturbation occurs so that the

functions are hopefully a good approximation of the true cost of the three important parameters - storage, access time, and update time.

The heuristic is presented in more detail below.

Repeat

For each access path i

Determine its current access method j

For every other access method k

If either the access or update time is expressed as a global constraint and these constraints are not violated then

Compute the conversion cost from access method j to access method k (This is a function of the file sizes which the access path relates).

Calculate the global objective cost factors

If storage requirement is a global objective then

Calculate the storage cost, arising from the conversion, as follows

Determine the marginal cost per storage unit by consulting the marginal storage cost function. This function is calculated at the end of the heuristic if any perturbation occurs, and has a negative cost if storage is saved.

Multiply this cost by the change in the number of storage units as a result of the conversion.

If access time is a global objective then

Calculate the access time cost, arising from the conversion, as follows

Determine the marginal cost per access time unit by consulting the marginal access time cost function

Multiply this cost by the forecasted access rate per unit time

Multiply this by the length of the planning horizon (the planning horizon is set at generation time and its value affects the volatility of the corresponding database system).

If update time is a global objective then

Calculate the update time cost, arising from the conversion, as follows

Determine the marginal cost per update time unit by consulting the marginal update time cost function

Multiply this cost by the forecasted update rate per unit time

Multiply this by the length of the planning horizon

Weight the global objective cost factors according to the trade-off functions existing between the objectives, add the weighted cost factors, and normalize the result so that the contribution of these cost factors is proportional to the number of global objectives

Add the conversion cost to get the resultant cost

Find the best access method to convert to

If the resultant cost is negative and less than the current minimum value then

Remember the access method and the new minimum value

Calculate the per unit costs for the global constraints and objectives

If storage is relinquished then

Divide the resultant cost by the number of storage units saved and remember it

If access time is relinquished by the conversion and access time is a global objective then

Divide the resultant cost by the number of access time units saved and remember it

If update time is relinquished by the conversion and update time is a global objective then

Divide the resultant cost by the number of update time units saved and remember it

Update the cost tables for generation of the marginal cost function

If storage is a global constraint then

Select the conversion that results in the lowest storage cost per unit

If this cost is one of the lowest ten found so far by the heuristic then save it in the GIVESTORAGE cost table. (If ten elements are already in the table, overwrite the highest element)

If storage is a global objective then

Select the conversion that results in the highest storage cost per unit

If this cost is one of the highest ten found so far by the heuristic then save it in the TAKESTORAGE cost table. (If ten elements are already in the table, overwrite the lowest element)

If access time is a global objective then

Select the conversion that results in the highest access time cost per unit

If this cost is one of the highest ten found so far by the heuristic then save it in the TAKEACCESSTIME cost table. (If ten elements are already in the table, overwrite the lowest element)

If update time is a global objective then

Select the conversion that results in the highest update time cost per unit

If this cost is one of the highest ten found so far by the heuristic then save it in the TAKEUPDATETIME cost table. (If ten elements are already in the table, overwrite the lowest element)

Decide if we are going to convert any access methods

If the minimum value is negative then

Assume that the conversion has taken place

While the proposed solution is still infeasible (storage constraint violated), the value is still negative, and there are still unused elements in the GIVESTORGAE list,

Add the value of the element with the lowest marginal storage cost of the unused elements of the GIVESTORAGE list to the solution (In effect this downgrades another access path so the value will normally be positive)

If all of the GIVESTORAGE list has been used up then reiterate to generate the next ten minimum values

If the resulting value is still negative, perform the perturbations (make all the required access path changes)

until no perturbations occur

Recompute the marginal cost functions

If storage is a global objective then

Recompute the marginal storage cost function as follows

The price of additional storage is given by the negative of the lowest value in the TAKESTORAGE list

The price for removal of storage up to the capacity given by the sum available from the ten elements of the TAKESTORAGE list is linearly interpolated between these ten points, from the lowest to the highest valued element

The price of removing further storage than the maximum available in the TAKESTORAGE list is given by the value of the highest element in the TAKESTORAGE list

If access time is a global objective then

Recompute the marginal access time cost function as follows

The price of additional access time is given by the negative of the lowest value in the TAKEACCESSTIME list

The price for removal of access time up to the capacity given by the sum available from the ten elements of the TAKEACCESSTIME list is linearly interpolated between these ten points, from the lowest to the highest valued element

The price of removing further access time than the maximum available in the TAKEACCESSTIME list is given by the value of the highest element in the TAKEACCESSTIME list

If update time is a global objective then

Recompute the marginal update time cost function as follows

The price of additional update time is given by the negative of the lowest value in the TAKEUPDATETIME list

The price for removal of update time up to the capacity given by the sum available from the ten elements of the TAKEUPDATETIME list is linearly interpolated between these ten points, from the lowest to the highest element

The price of removing further update time than the maximum available in the TAKEUPDATETIME list is given by the value of the highest element in the TAKEUPDATETIME list

References

1. ALBANO, A. and ORSINI, R. (1981) "An Interactive Integrated System to Design and use Data Bases". Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling; ACM SIGMOD, Volume 11, Number 2, pp 91-93.

2. ANSI/X3/SPARC (1975), Study Group on Data Base Management Systems, Interim Report ANSI 75-02-08, FDT, Bulletin of ACM - SIGMOD, Volume 7, Number 2.

3. BROWN, R.G. (1959) "Statistical Forecasting for Inventory Control". McGraw-Hill.

4. CHEN, P.P. (1976) "The Entity-Relationship Model - Towards a Unified View of Data". ACM Transactions on Database Systems, Volume 1, Number 1, pp 9-36.

5. DAELLENBACH, H.G. and GEORGE, J.A. (1978) "Introduction to Operations Research Techniques". Allyn and Bacon.

6. DEARNLEY, P.A. (1973) "Self Organising Data Management Systems". Ph.D. Thesis, University of East Anglia, Norwich, United Kingdom.

7. GEORGES, T.M. (1979) "A Course in Analytical Writing for Scientists and Engineers". NOAA, Wave Propagation Laboratory, Boulder, Colorado.

8. HAYES-ROTH, B. and HAYES-ROTH, F. (1975) "Plasticity in Memorial Networks". Journal of Verbal Learning and Verbal Behaviour, Volume 14, pp 506-522.

9. HENDRIX, G.G. (1981) "Mediating the Views of Databases and Database Users". Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling; ACM SIGMOD, Volume 11, Number 2, pp 131-132.

10. HOSKINS, P. and SIMPSON, G. (1981) "Burroughs LINC Advanced Information System". 7th Conference of the New Zealand Computer Society, Conference Papers, pp 402-411.

11. Infotech (1975), Database Systems, Infotech State of the Art Report, Infotech International, Berkshire.

12. KATZ, R.H. (1981) "Heterogeneous Databases and High Level Abstraction". Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling; ACM SIGMOD, Volume 11, Number 2, pp 135-137.

13. KNUTH, D.E. (1973) "The Art of Computer Programming", Volume 3, Sorting and Searching, Addison-Wesley.

14. KOLLIAS, J.G. (1976) "The Design of Data Base Management Systems using Linear Programming Techniques". Ph.D. Thesis, University of East Anglia, Norwich, United Kingdom.

15. MCLEOD, D. (1981) "On Conceptual Database Modelling". Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling; ACM SIGMOD, Volume 11, Number 2, pp 161-163.

16. PRICE, C.E. (1971) "Table Lookup Techniques", ACM Computing Surveys, Volume 3, pp 56-58.

17. RIDWAN, M.M. and LOY, Z. (1978) "A Data Abstraction Model". Technical Report 78/1, Department of Computer Science, University of Canterbury, Christchurch, New Zealand.

18. RIDWAN, M.M. (1979) "Abstractions in Data Base System Architecture". Ph.D. Thesis, University of Canterbury, Christchurch, New Zealand.

19. SENKO, M.E. (1976) "DIAM II; The Binary Infological Level and its Database Language - FORAL". Proceedings of Conference of Data: Abstraction, Definition and Structure; ACM SIGPLAN, Volume 8, Number 2, pp 121-140.

20. STOCKER, P.M. and DEARNLEY, P.A. (1974) "A Self-Organising Data Base Management System". Data Base Management, edited by Klimbie, J.W. and Koffeman, K.L., North Holland.

21. SU, S.Y.W.; LAM, H. and LO, D.H. (1981) "Transformation of Data Traversals and Operations in Applications Programs to Account for Semantic Changes of Databases". ACM Transactions on Database Systems, Volume 6, Number 2, pp 255-294.

22. SULLIVAN, W.G. and CLAYCOMBE, W.W. (1977) "Fundamentals of Forecasting". Reston.

23. WIEDERHOLD, G.; KAPLAN, J. and SAGALOWICZ, D. (1981) "Research in Knowledge Base Management Systems". ACM SIGMOD Record, Volume 11, Number 3, pp 26-54.